In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import csv
import random as rand
import math
import operator
from tqdm import tqdm_notebook as tqdm
```

Working on Dataset

In [2]:
```
df = pd.read_csv('football.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	name	club	age	position	position_cat	market_value	page_views	fpl_value	fpl_sel	fpl_points
0	Alexis Sanchez	Arsenal	28	LW	1	65.0	4329	12.0	17.10%	264
1	Mesut Ozil	Arsenal	28	AM	1	50.0	4395	9.5	5.60%	167
2	Petr Cech	Arsenal	35	GK	4	7.0	1529	5.5	5.90%	134
3	Theo Walcott	Arsenal	28	RW	1	20.0	2393	7.5	1.50%	122
4	Laurent Koscielny	Arsenal	31	CB	3	22.0	912	6.0	0.70%	121

In [4]:
```
#changing fpl_sel to float
fpl_sel = df['fpl_sel'].copy()
for i in range(len(fpl_sel)):
    fpl_sel[i] = float(fpl_sel[i][:-1])
df['fpl_sel']=fpl_sel
```

In [5]:
```
df=df.drop(columns=['name'])
df.head()
```

Out[5]:

	club	age	position	position_cat	market_value	page_views	fpl_value	fpl_sel	fpl_points	region
0	Arsenal	28	LW	1	65.0	4329	12.0	17.1	264	3.0
1	Arsenal	28	AM	1	50.0	4395	9.5	5.6	167	2.0
2	Arsenal	35	GK	4	7.0	1529	5.5	5.9	134	2.0
3	Arsenal	28	RW	1	20.0	2393	7.5	1.5	122	1.0
4	Arsenal	31	CB	3	22.0	912	6.0	0.7	121	2.0

In [6]:
```
#converting categorical variables to numerical values
from sklearn import preprocessing
encoder = preprocessing.LabelEncoder()
df['club']=encoder.fit_transform(df['club'])
df['position']=encoder.fit_transform(df['position'])
df['nationality']=encoder.fit_transform(df['nationality'])
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
club            0
age             0
position        0
position_cat    0
market_value    0
page_views      0
fpl_value       0
fpl_sel         0
fpl_points      0
region          1
nationality     0
new_foreign     0
age_cat         0
club_id         0
big_club        0
new_signing     0
dtype: int64
```

In [8]:
```
df = pd.get_dummies(df, columns=["region"],drop_first=True)
```

In [9]:
```
df.head()
```

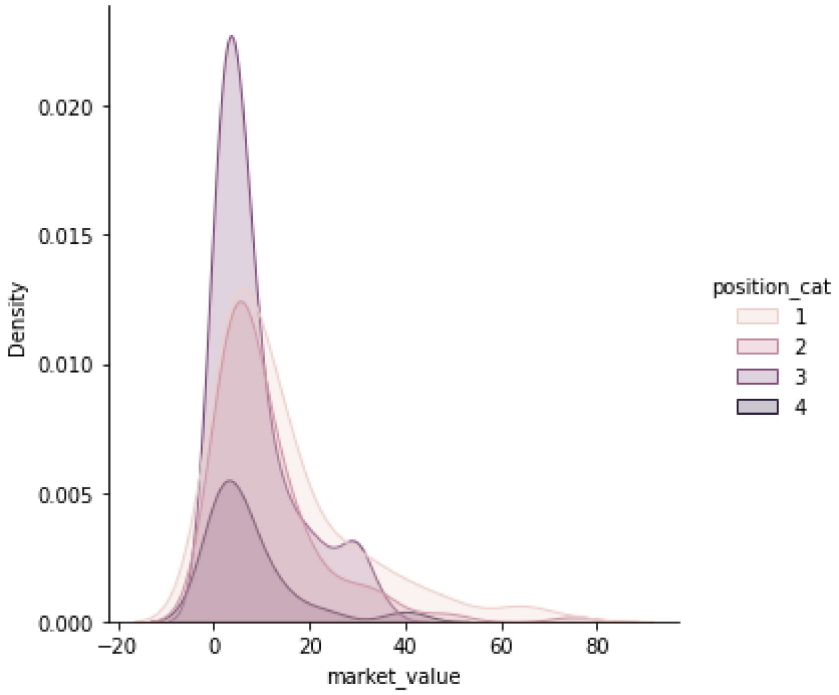
Out[9]:

	club	age	position	position_cat	market_value	page_views	fpl_value	fpl_sel	fpl_points	nationality
0	0	28	8	1	65.0	4329	12.0	17.1	264	12
1	0	28	0	1	50.0	4395	9.5	5.6	167	26
2	0	35	5	4	7.0	1529	5.5	5.9	134	18
3	0	28	11	1	20.0	2393	7.5	1.5	122	22
4	0	31	1	3	22.0	912	6.0	0.7	121	25

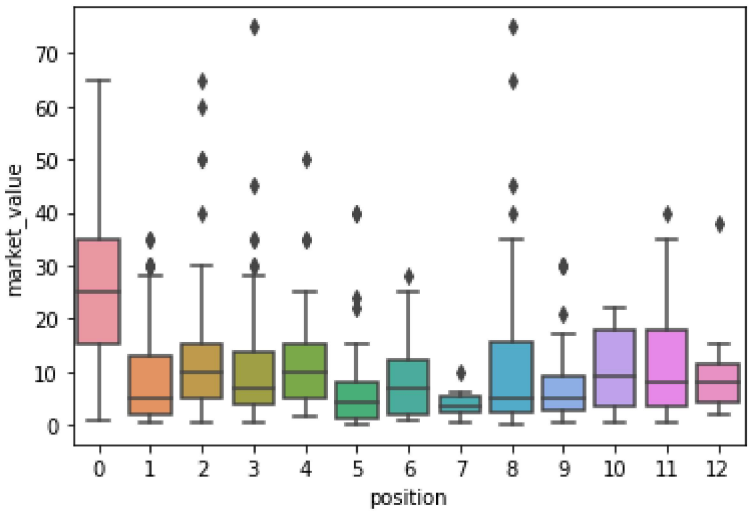
In [10]:
```
df.dtypes
```

Out[10]:
```
club              int32
age               int64
position          int32
position_cat      int64
market_value    float64
page_views        int64
fpl_value       float64
fpl_sel          object
fpl_points        int64
nationality       int32
new_foreign       int64
age_cat           int64
club_id           int64
big_club          int64
new_signing       int64
region_2.0        uint8
region_3.0        uint8
region_4.0        uint8
dtype: object
```

In [11]:
```
sns.displot(df,x='market_value',hue="position_cat",kind='kde',fill=True)
plt.show()
```



In [12]:
```
sns.boxplot(x='position',y='market_value',data=df)
plt.show()
```



In [13]:
```
df.describe()
```

Out[13]:

	club	age	position	position_cat	market_value	page_views	fpl_value	fpl_poi
count	461.000000	461.000000	461.000000	461.000000	461.000000	461.000000	461.000000	461.000(
mean	9.334056	26.804772	4.546638	2.180043	11.012039	763.776573	5.447939	57.3145
std	5.726475	3.961892	3.323908	1.000061	12.257403	931.805757	1.346695	53.113(
min	0.000000	17.000000	0.000000	1.000000	0.050000	3.000000	4.000000	0.000(
25%	5.000000	24.000000	2.000000	1.000000	3.000000	220.000000	4.500000	5.000(
50%	9.000000	27.000000	4.000000	2.000000	7.000000	460.000000	5.000000	51.000(
75%	14.000000	30.000000	7.000000	3.000000	15.000000	896.000000	5.500000	94.000(
max	19.000000	38.000000	12.000000	4.000000	75.000000	7664.000000	12.500000	264.000(

In [14]:
```
#standardizing data to remove outliers influence
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
new_df = scaler.fit_transform(df)
new_df = pd.DataFrame(df,columns=list(df.columns))
df.head()
```

Out[14]:

	club	age	position	position_cat	market_value	page_views	fpl_value	fpl_sel	fpl_points	nationality
0	0	28	8	1	65.0	4329	12.0	17.1	264	12
1	0	28	0	1	50.0	4395	9.5	5.6	167	26
2	0	35	5	4	7.0	1529	5.5	5.9	134	18
3	0	28	11	1	20.0	2393	7.5	1.5	122	22
4	0	31	1	3	22.0	912	6.0	0.7	121	25

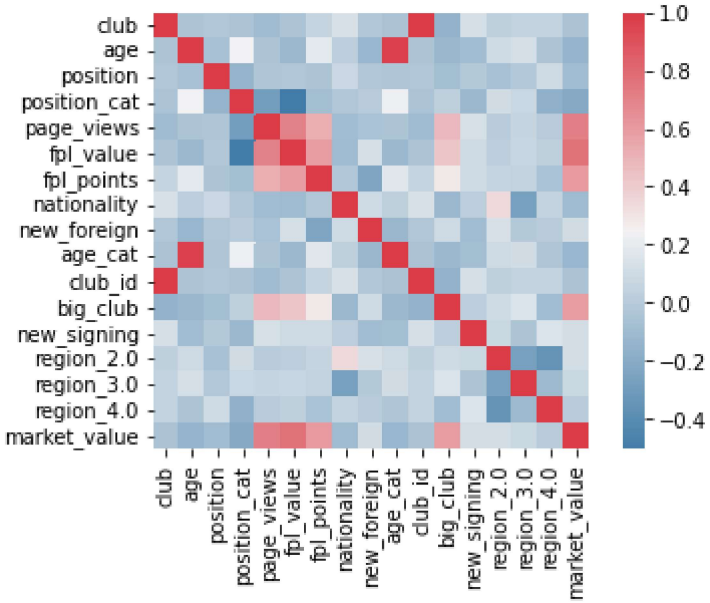
In [15]:
```
#splitting dataset to train and test
from sklearn.model_selection import train_test_split
market_value=new_df['market_value'].copy()
x = new_df.drop(columns=['market_value'])
y = market_value
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.20,random_state=42)
```

In [16]:
```
train_df = X_train.copy()
train_df['market_value']=y_train
train_df.head()
```

Out[16]:

	club	age	position	position_cat	page_views	fpl_value	fpl_sel	fpl_points	nationality	new_foreign
410	17	26	1	3	177	4.5	0.2	35	5	0
265	11	29	0	1	2280	7.0	3.5	102	51	0
57	2	36	9	1	215	4.5	0.4	0	1	0
199	8	33	1	3	649	4.5	3.3	63	33	0
175	7	21	3	2	157	4.5	0.1	0	19	0

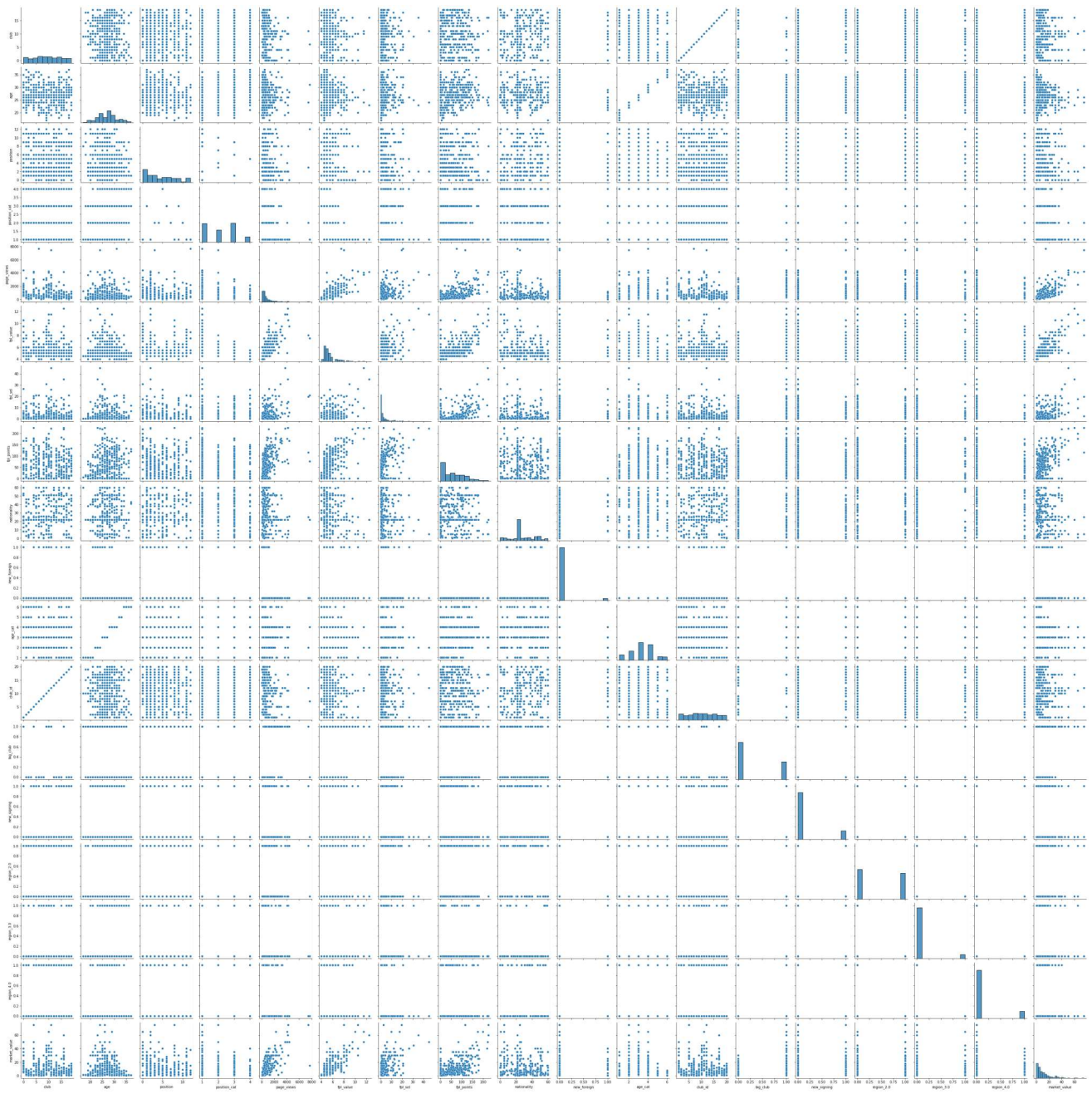
In [17]:
```
corr = train_df.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(240,10,as_cmap=True),
            square=True)
plt.show()
```



In [18]:
```
corr['market_value'].sort_values()
```

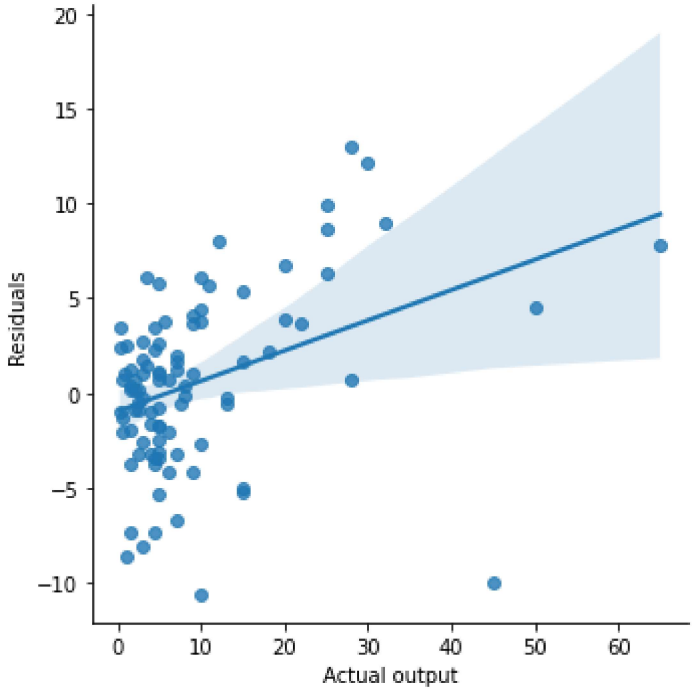
Out[18]:
```
position_cat   -0.210206
age            -0.139648
age_cat        -0.134056
nationality    -0.101599
position       -0.090664
club           -0.051572
club_id        -0.051572
region_4.0      0.006618
region_3.0      0.078955
new_foreign     0.107322
region_2.0      0.115559
new_signing     0.116797
big_club        0.593065
fpl_points      0.600608
page_views      0.713693
fpl_value       0.772383
market_value    1.000000
Name: market_value, dtype: float64
```

In [29]:
```
sns.pairplot(train_df,height=2.5)
plt.show()
```

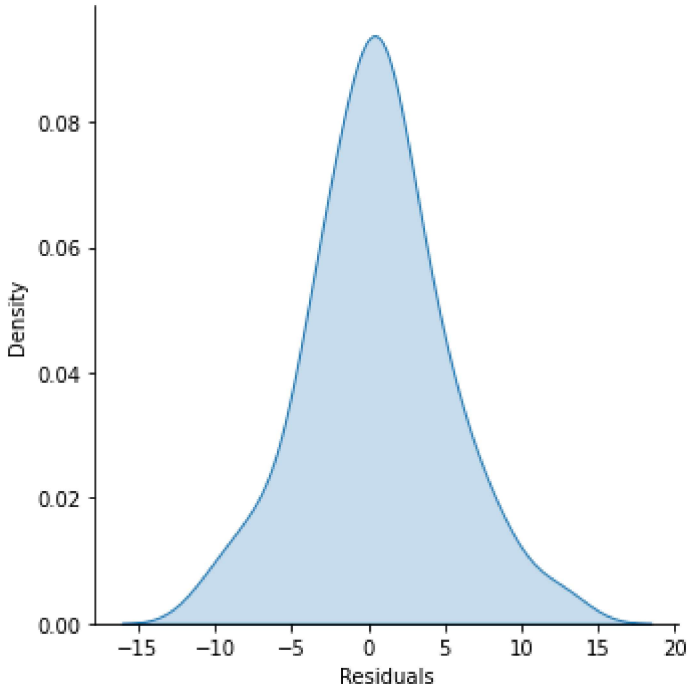


Regression Models

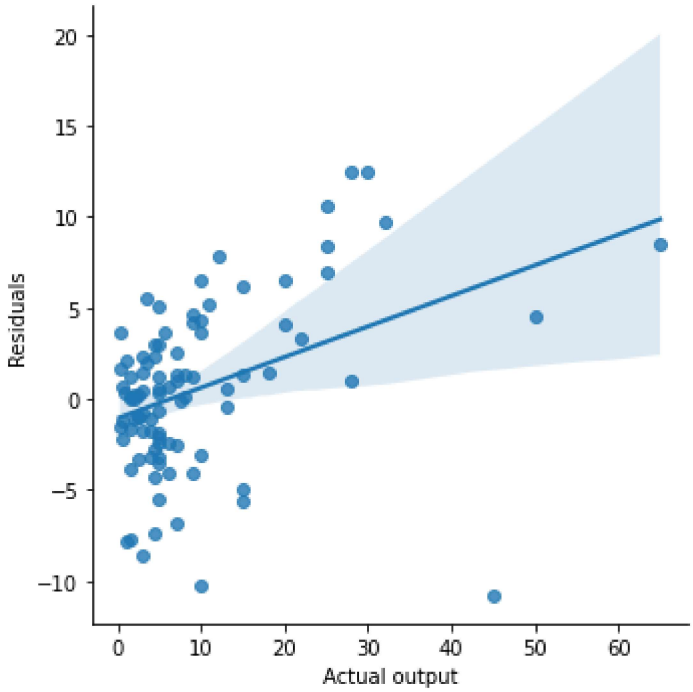
In [27]:
```
#Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score as r2,mean_squared_error as mse
model = LinearRegression()
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual
output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



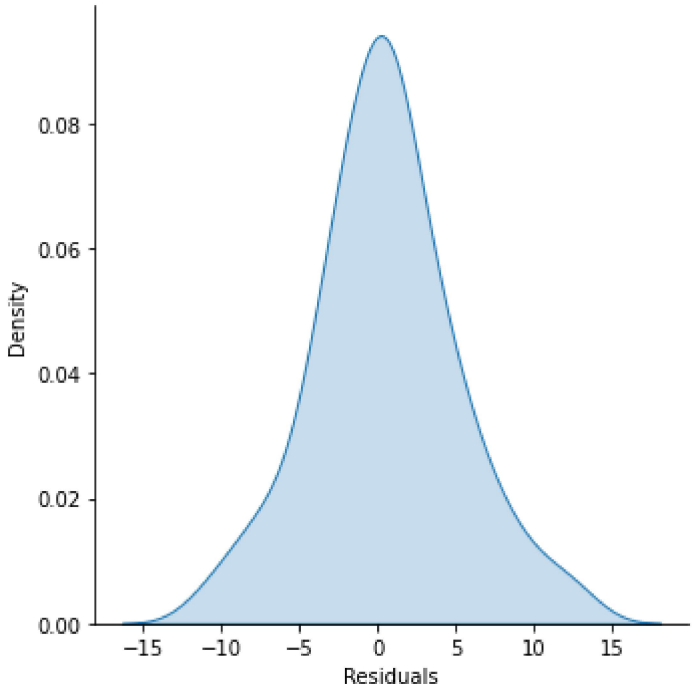

```
R2 Score:  0.8294659156645359
RMSE value:  4.543505012003856
Training R2 Score:  0.7671206543211353
```



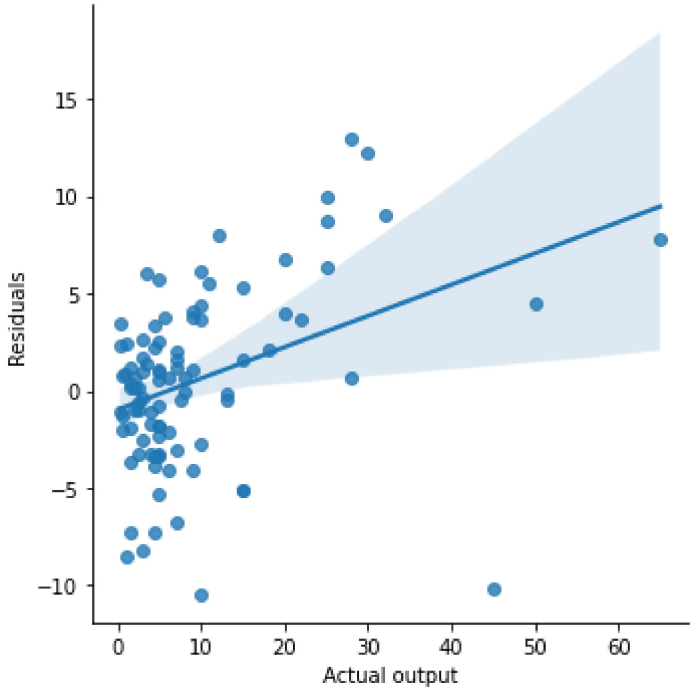
In [21]:
```
#Lasso Regression
from sklearn.linear_model import Lasso
model = Lasso(alpha=0.05)
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual
output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



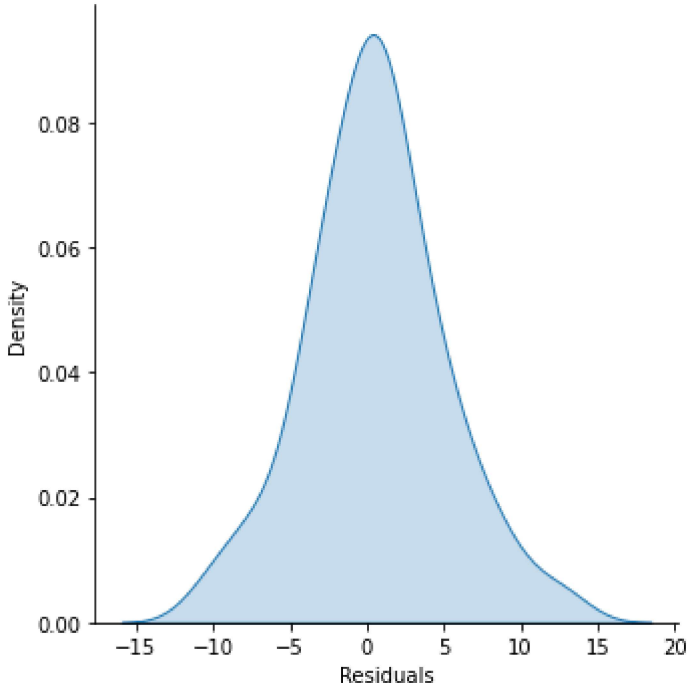

```
R2 Score:  0.8247780972295424
RMSE value:  4.605529947993149
Training R2 Score:  0.7657064325979929
```



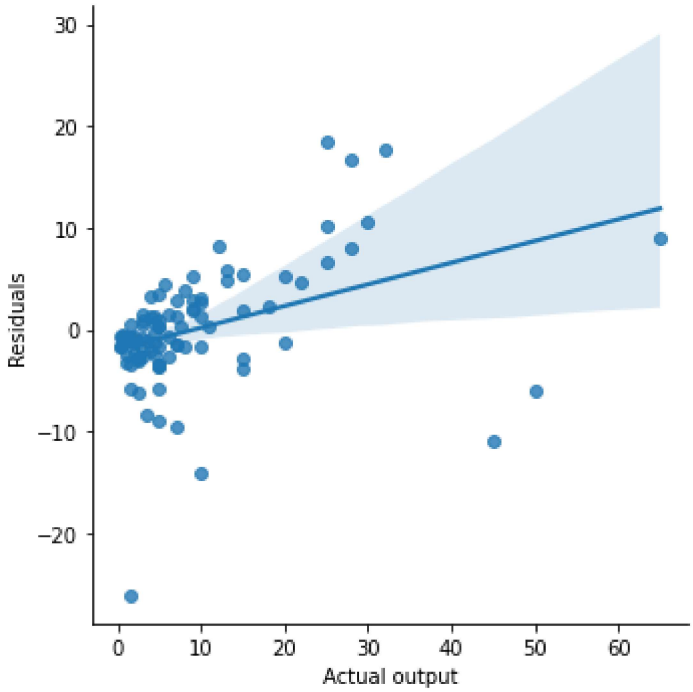
In [22]:
```
#Ridge Regression
from sklearn.linear_model import Ridge
model = Ridge()
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual
output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



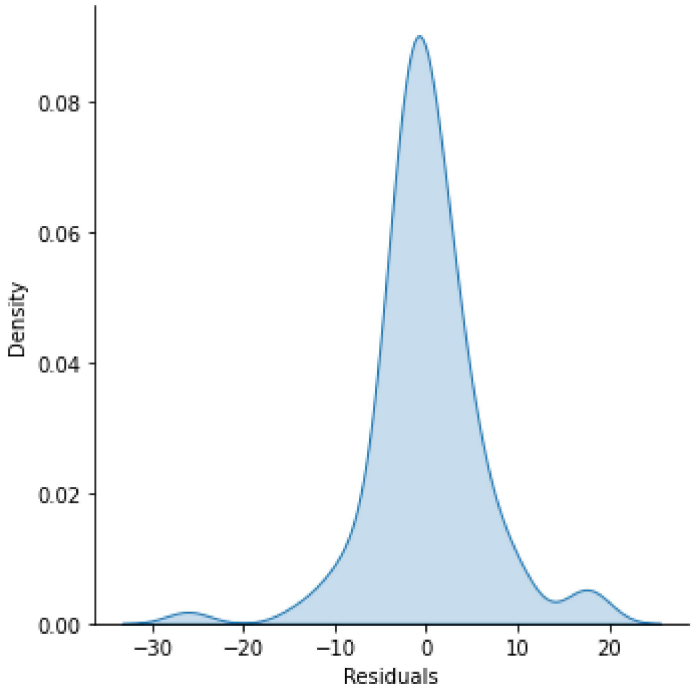

```
R2 Score:  0.829680149285458
RMSE value:  4.540650223689509
Training R2 Score:  0.7670797372595546
```



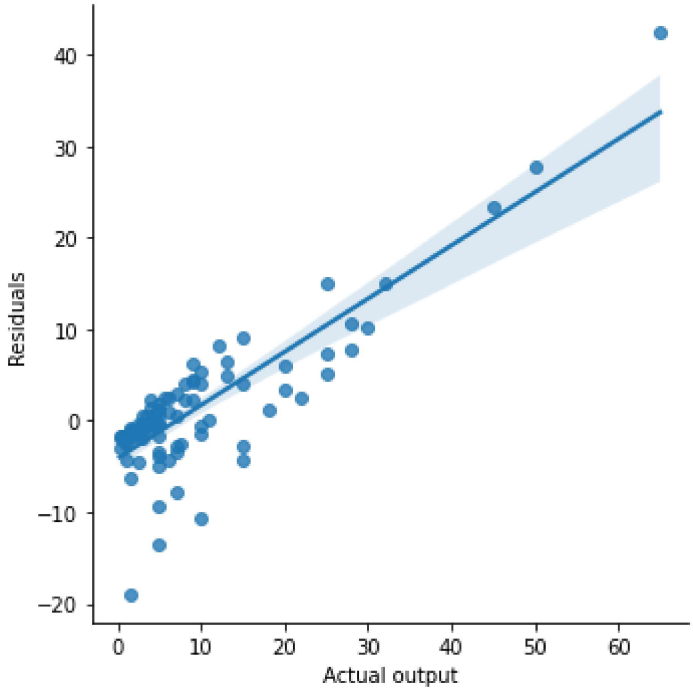
In [23]:
```
#KNN Regression
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=5)
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual
output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



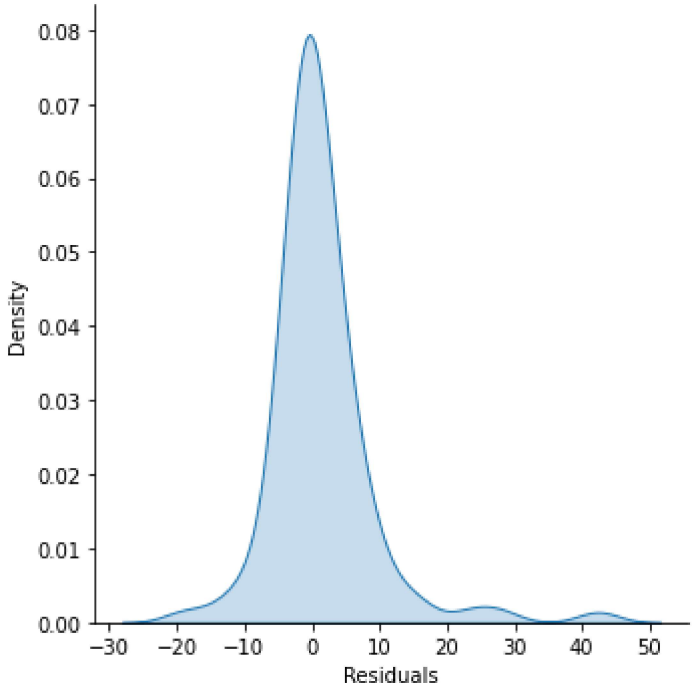

```
R2 Score:  0.7096608105629928
RMSE value:  5.928410184174839
Training R2 Score:  0.7012243403050928
```



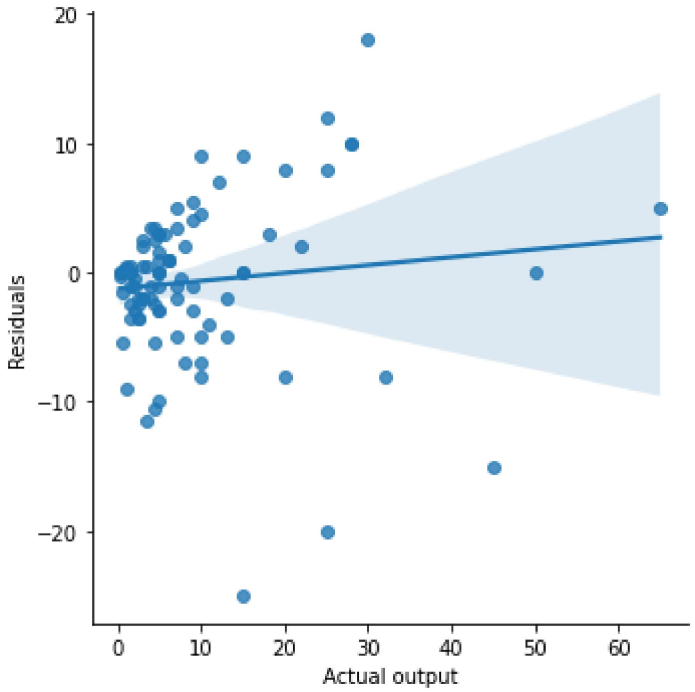
In [24]:
```
#SVM Regression
from sklearn.svm import SVR
model = SVR()
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual
output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



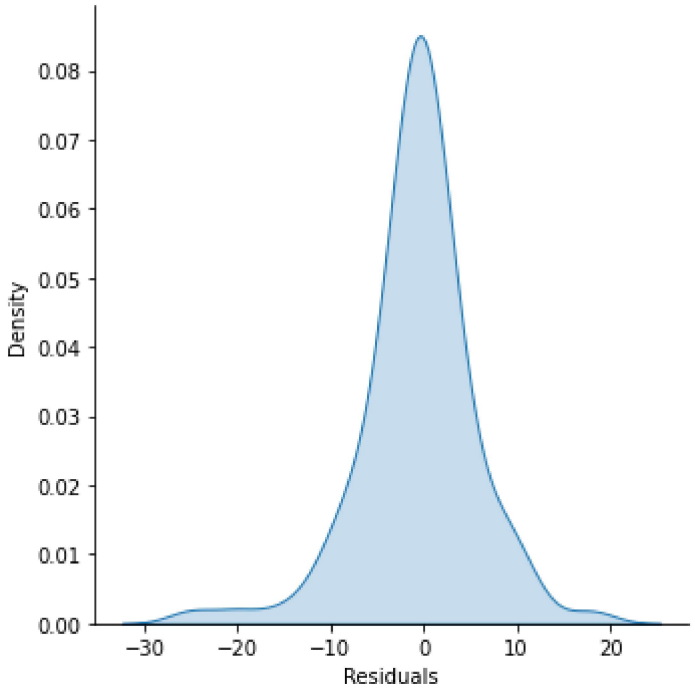

```
R2 Score:  0.5149547103238603
RMSE value:  7.662607147378696
Training R2 Score:  0.43536455971169474
```



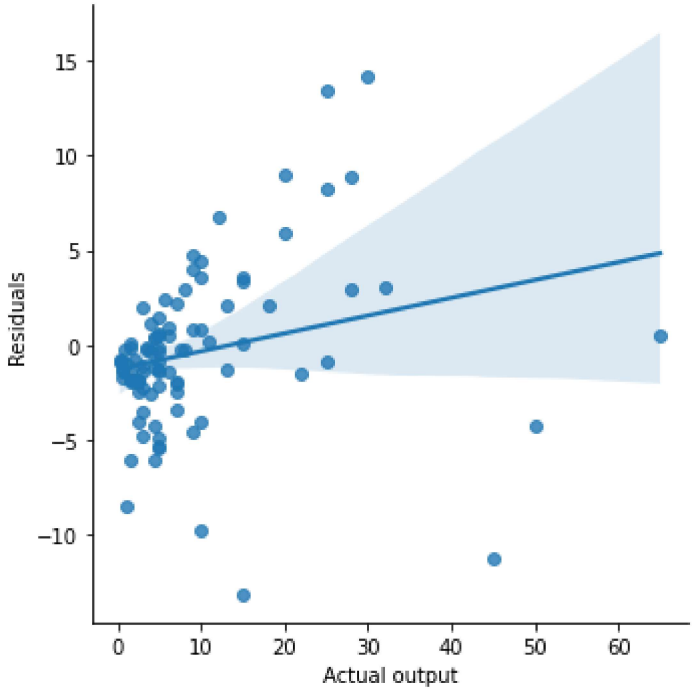
In [25]:
```
#Tree Regression
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor()
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual
output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



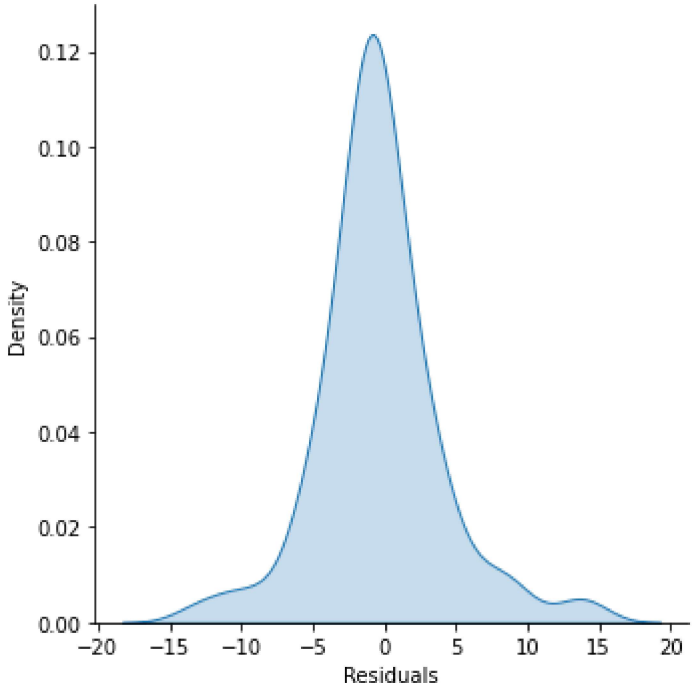

```
R2 Score:  0.691647301370805
RMSE value:  6.109550957576654
Training R2 Score:  1.0
```



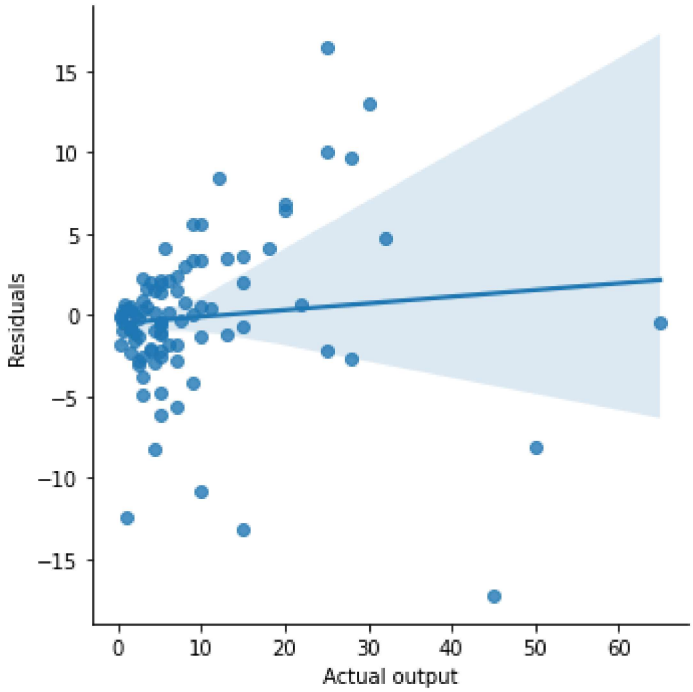
In [28]:
```
#Random forest regression
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual
output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



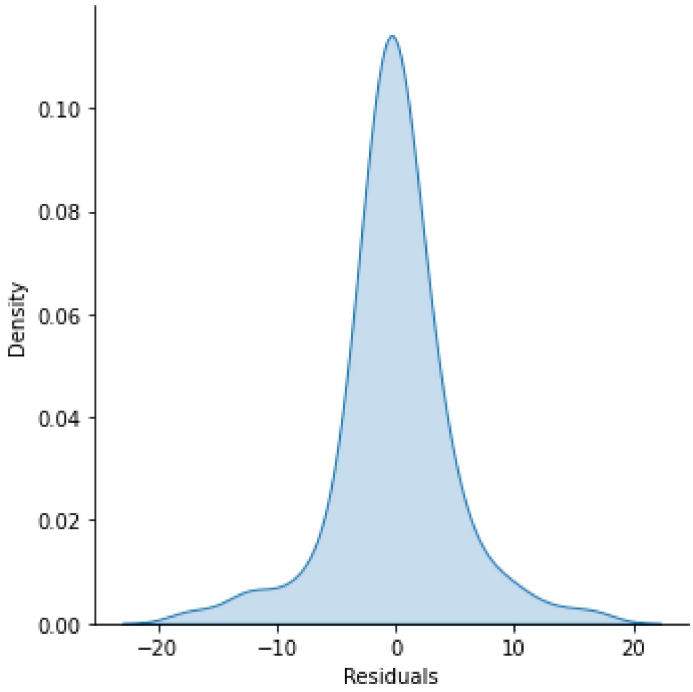

```
R2 Score:  0.8501820543594122
RMSE value:  4.258604742734257
Training R2 Score:  0.967530200965403
```



In [27]:
```
#Gradient Boosted Regression
from sklearn.ensemble import GradientBoostingRegressor
model = GradientBoostingRegressor()
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual
output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```




```
R2 Score:  0.8102704458837044
RMSE value:  4.792398324757715
Training R2 Score:  0.9707302656471123
```

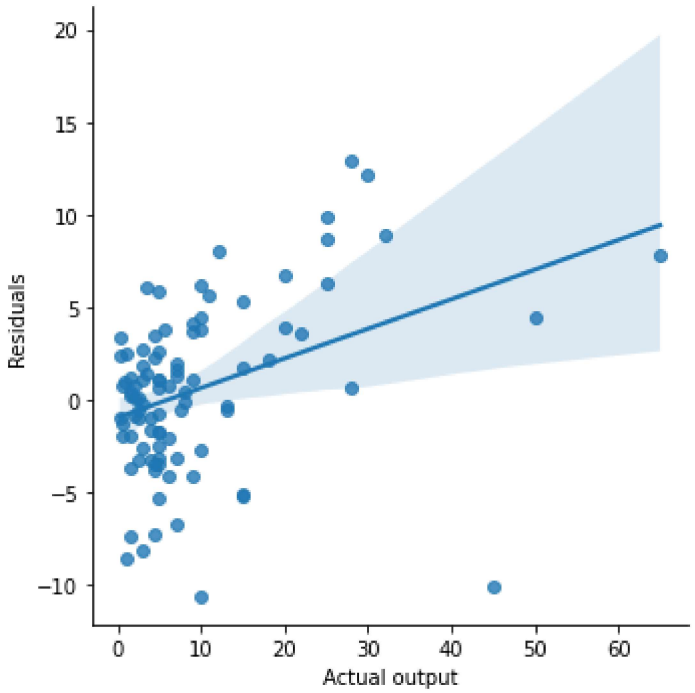


HyperParameters Tuning

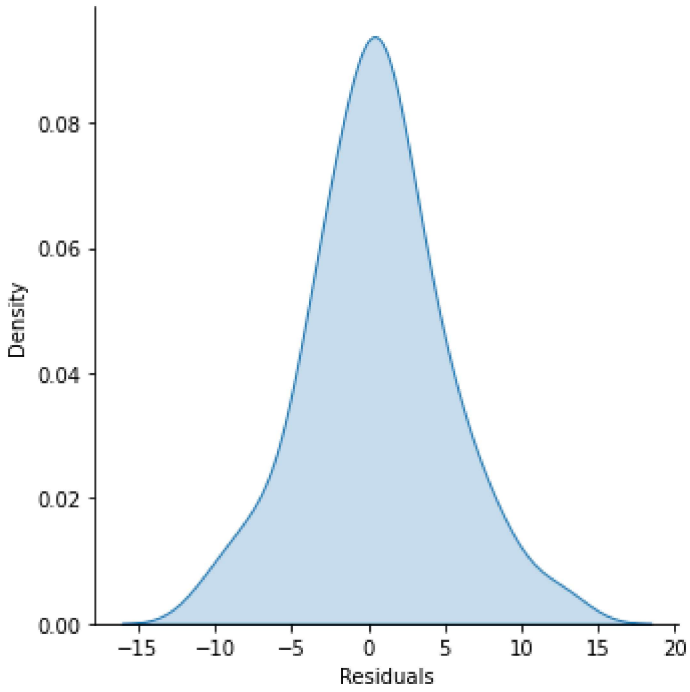
In [28]:
```
#For Linear Regression
from sklearn.model_selection import GridSearchCV,RepeatedKFold,RandomizedSearchCV
cv = RepeatedKFold(10,3,2)
estimator = LinearRegression()
parameters={
    "copy_X":[True],
    "fit_intercept":[True,False],
    "normalize":[True,False],
}
gridsearch = GridSearchCV(estimator,parameters,cv=cv)
model=gridsearch.fit(X_train,y_train)
bestparams = model.best_params_
print("Best Parameters: ",bestparams)
```

```
Best Parameters:  {'copy_X': True, 'fit_intercept': False, 'normalize': True}
```

In [29]:
```
model = LinearRegression().set_params(**bestparams)
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



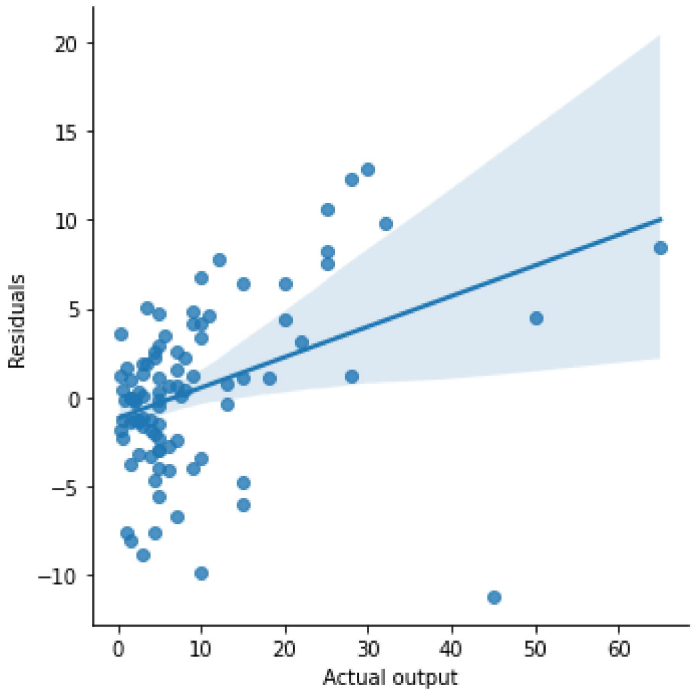

```
R2 Score:  0.8294659156645313
RMSE value:  4.543505012003916
Training R2 Score:  0.7671206543211353
```



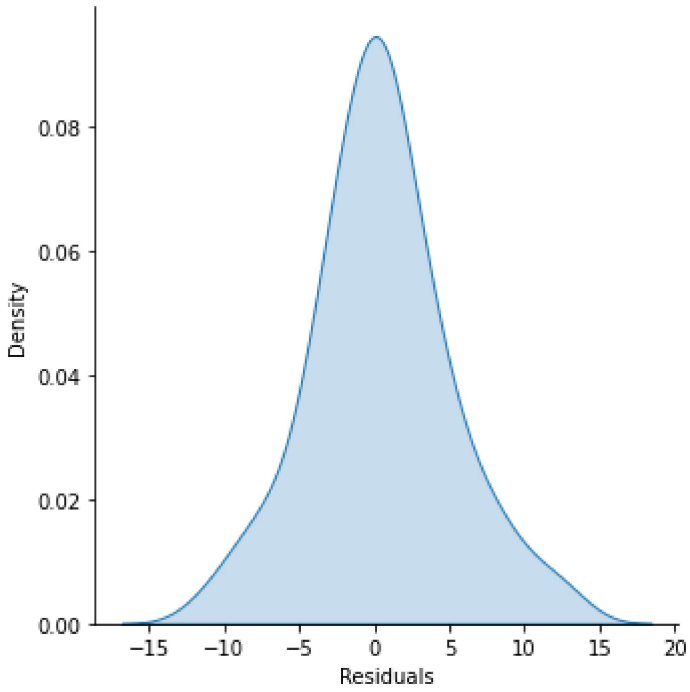
In [30]:
```
#For Lasso Regression
cv = RepeatedKFold(10,3,2)
estimator = Lasso()
parameters={
    "alpha":np.linspace(0,1,11)[1:],
    "max_iter":[10**i for i in range(3,7)],
    "normalize":[True,False],
    "tol":[10**(-i) for i in range(1,5)],
    "positive":[True,False],
    "selection":['cyclic','random']
}
gridsearch = GridSearchCV(estimator,parameters,cv=cv)
model=gridsearch.fit(X_train,y_train)
bestparams = model.best_params_
print("Best Parameters: ",bestparams)
```

```
Best Parameters:  {'alpha': 0.1, 'max_iter': 1000, 'normalize': False, 'positive': F
alse, 'selection': 'random', 'tol': 0.1}
```

In [31]:
```
model = Lasso().set_params(**bestparams)
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



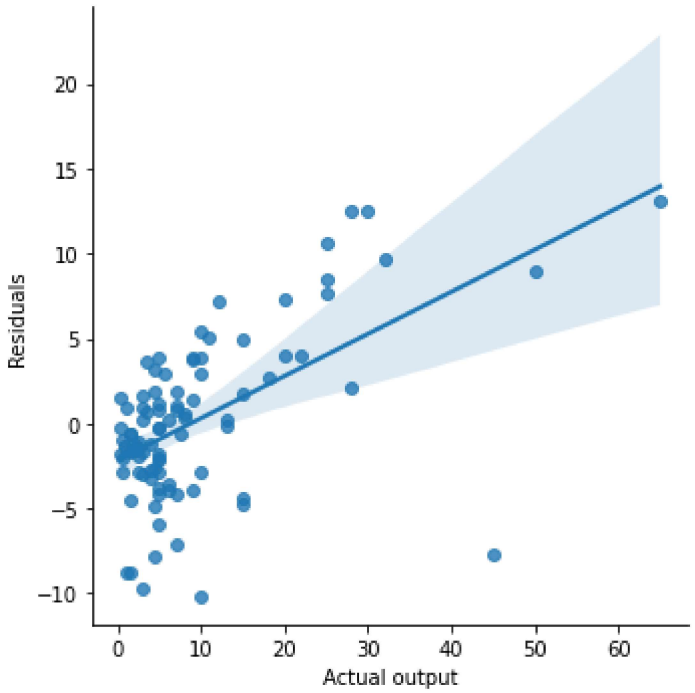

```
R2 Score:  0.8238851435003333
RMSE value:  4.617250226376804
Training R2 Score:  0.7628115339552893
```



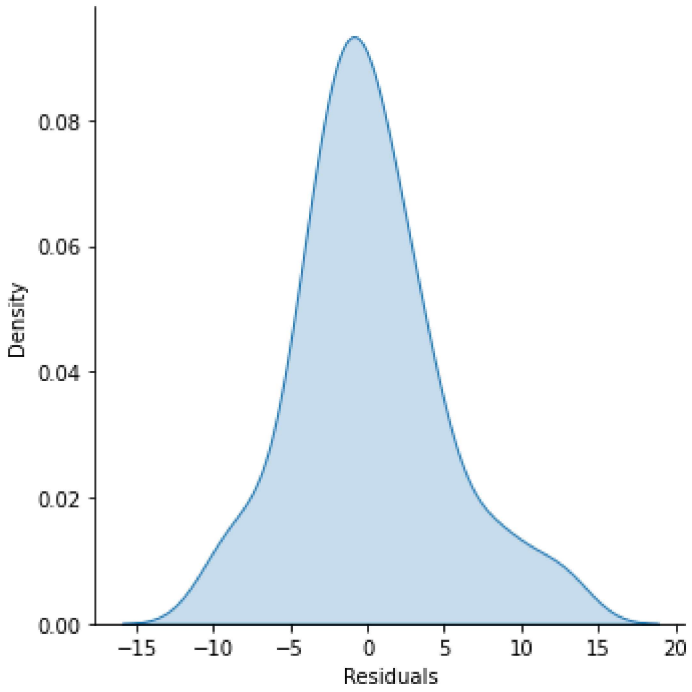
In [32]:
```
#For Ridge Regression
cv = RepeatedKFold(10,3,2)
estimator = Ridge()
parameters={
    "alpha":np.linspace(0,1,11)[1:],
    "max_iter":[10**i for i in range(1,7)],
    "normalize":[True,False],
    "fit_intercept":[True,False],
    "copy_X":[True]
}
gridsearch = GridSearchCV(estimator,parameters,cv=cv)
model=gridsearch.fit(X_train,y_train)

bestparams = model.best_params_
print("Best Parameters: ",bestparams)
```

```
Best Parameters:  {'alpha': 0.30000000000000004, 'copy_X': True, 'fit_intercept': Tr
ue, 'max_iter': 10, 'normalize': True}
```

In [33]:
```
model = Ridge().set_params(**bestparams)
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



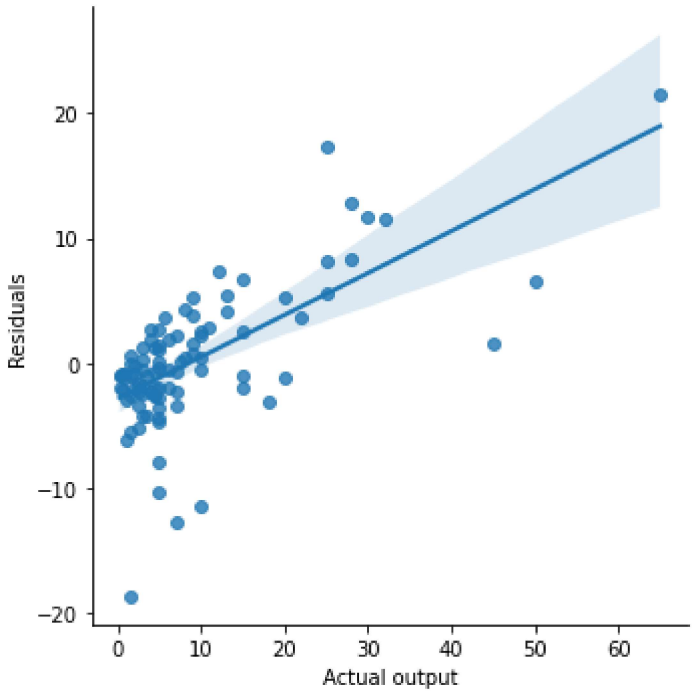

```
R2 Score:  0.8131010371606275
RMSE value:  4.756514886682781
Training R2 Score:  0.7564722263086593
```



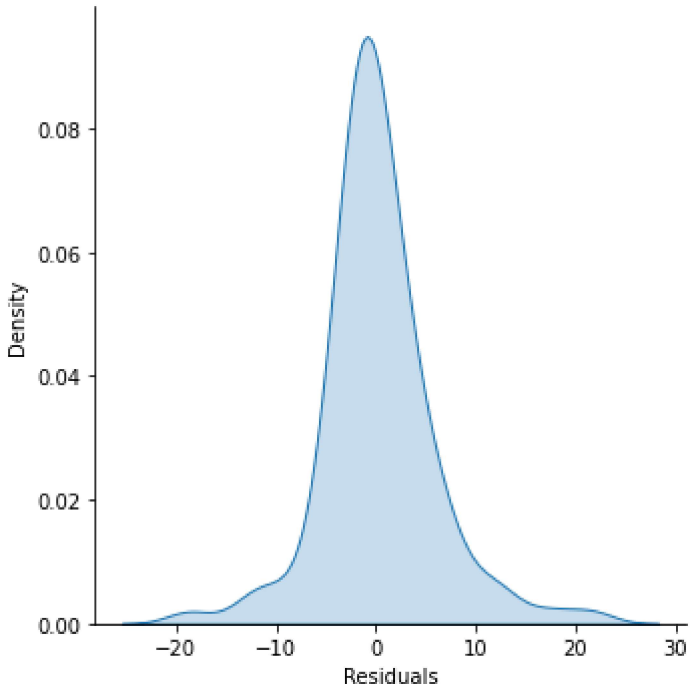
In [34]:
```
#For KNN Regression
cv = RepeatedKFold(10,3,2)
estimator = KNeighborsRegressor()
parameters={
    "n_neighbors":range(5,15),
    "p":[1,2,3,4,5],
    "weights":['uniform','distance']
}
gridsearch = GridSearchCV(estimator,parameters,cv=cv)
model=gridsearch.fit(X_train,y_train)

bestparams = model.best_params_
print("Best Parameters: ",bestparams)
```

```
Best Parameters:  {'n_neighbors': 11, 'p': 1, 'weights': 'uniform'}
```

In [35]:
```
model = KNeighborsRegressor().set_params(**bestparams)
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



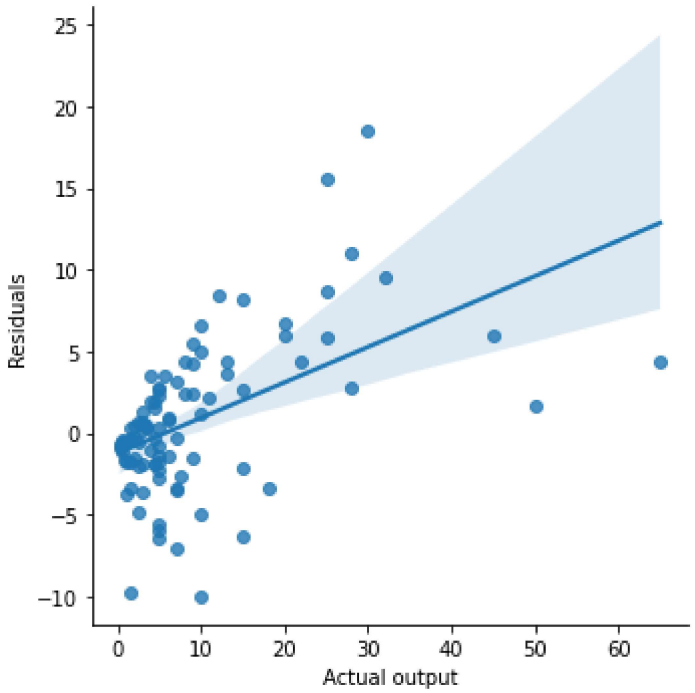

```
R2 Score:  0.7450892956602855
RMSE value:  5.554941006917307
Training R2 Score:  0.65048224037684
```



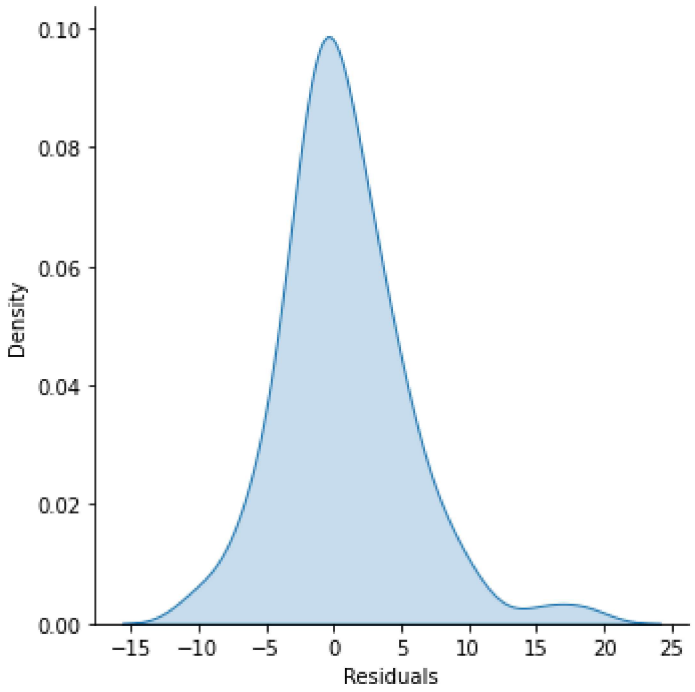
In [36]:
```
#For SVM Regression
cv = RepeatedKFold(10,3,2)
estimator = SVR()
parameters={
    "kernel":['poly', 'rbf','sigmoid'],
    "C":[100,1000,10000],
    "epsilon":[0.01,0.001,0.0001]
}
gridsearch = RandomizedSearchCV(estimator,parameters,scoring='r2')
model=gridsearch.fit(X_train,y_train)

bestparams = model.best_params_
print("Best Parameters: ",bestparams)
```

```
Best Parameters:  {'kernel': 'rbf', 'epsilon': 0.01, 'C': 1000}
```

In [37]:
```
model = SVR().set_params(**bestparams)
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



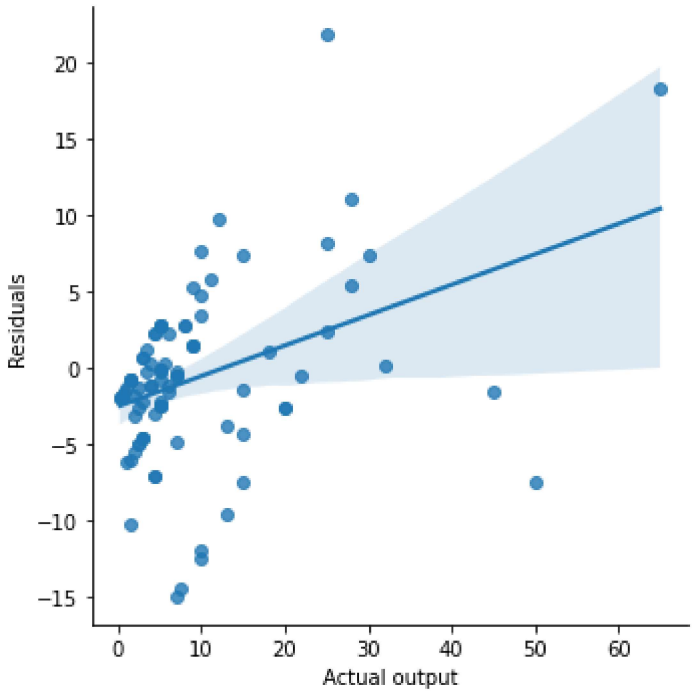

```
R2 Score:  0.8179350186586989
RMSE value:  4.694600331743394
Training R2 Score:  0.7078509092084575
```



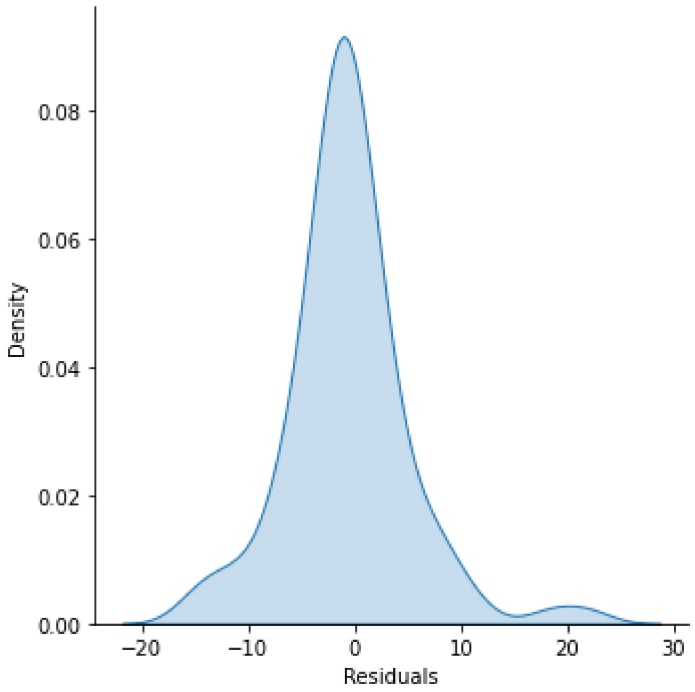
In [38]:
```
#For Tree Regression
cv = RepeatedKFold(10,3,2)
estimator = DecisionTreeRegressor()
parameters={
    "criterion":["mse", "friedman_mse", "mae"],
    "splitter":["best", "random"],
    "max_depth":[None,1,2,3,4,5],
    "max_features":[None,"auto", "sqrt", "log2"]
}
gridsearch = GridSearchCV(estimator,parameters,scoring='r2',cv=cv)
model=gridsearch.fit(X_train,y_train)

bestparams = model.best_params_
print("Best Parameters: ",bestparams)
```

```
Best Parameters:  {'criterion': 'mse', 'max_depth': 5, 'max_features': 'auto', 'spli
tter': 'random'}
```

In [39]:
```
model = DecisionTreeRegressor().set_params(**bestparams)
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



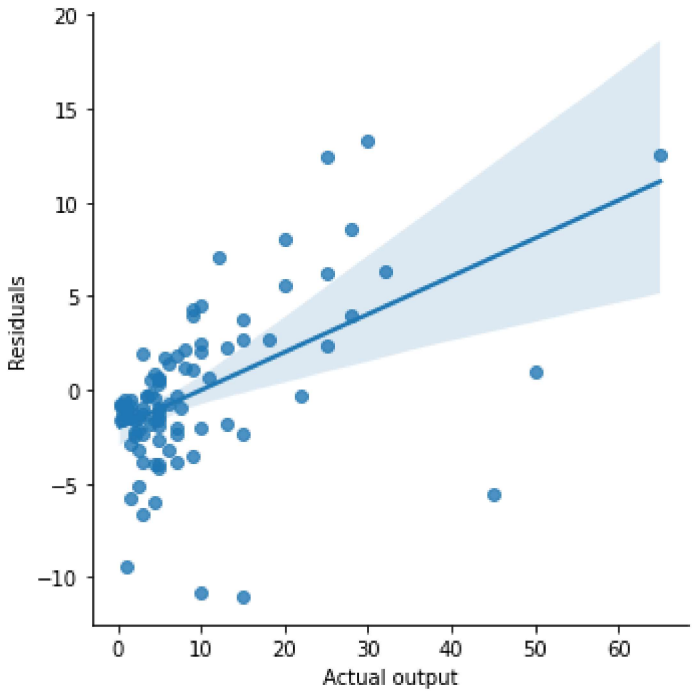

```
R2 Score:  0.7325192317706685
RMSE value:  5.690254561131599
Training R2 Score:  0.8330799241893435
```



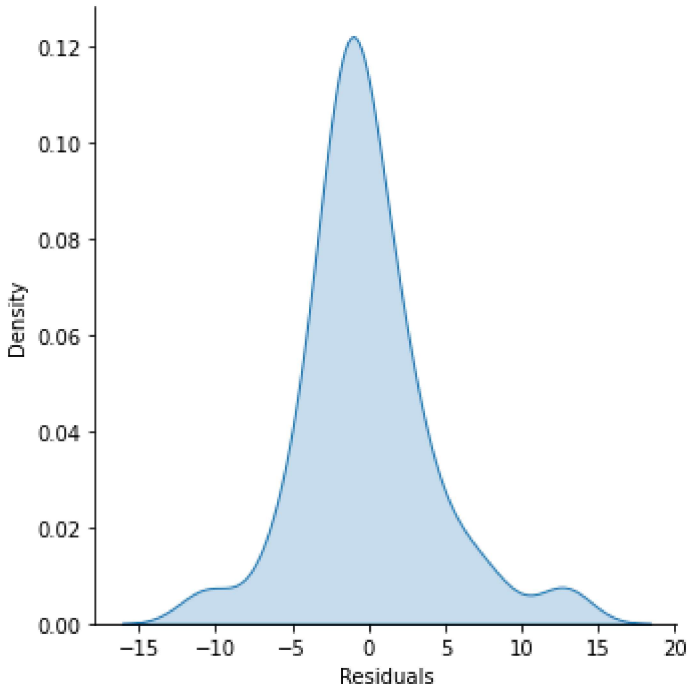
In [40]:
```
#For RandomForest Regression
cv = RepeatedKFold(10,3,2)
estimator = RandomForestRegressor()
parameters={
     "n_estimators":[10,100,200],
     "criterion":["mse","mae"],
     "max_depth":[None,5,10],
     "max_features":[None,"auto", "sqrt", "log2"],
}
gridsearch = RandomizedSearchCV(estimator,parameters,scoring='r2',cv=cv)
model=gridsearch.fit(X_train,y_train)

bestparams = model.best_params_
print("Best Parameters: ",bestparams)
```

```
Best Parameters:  {'n_estimators': 200, 'max_features': 'sqrt', 'max_depth': None,
'criterion': 'mae'}
```

In [41]:
```
model = RandomForestRegressor().set_params(**bestparams)
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```



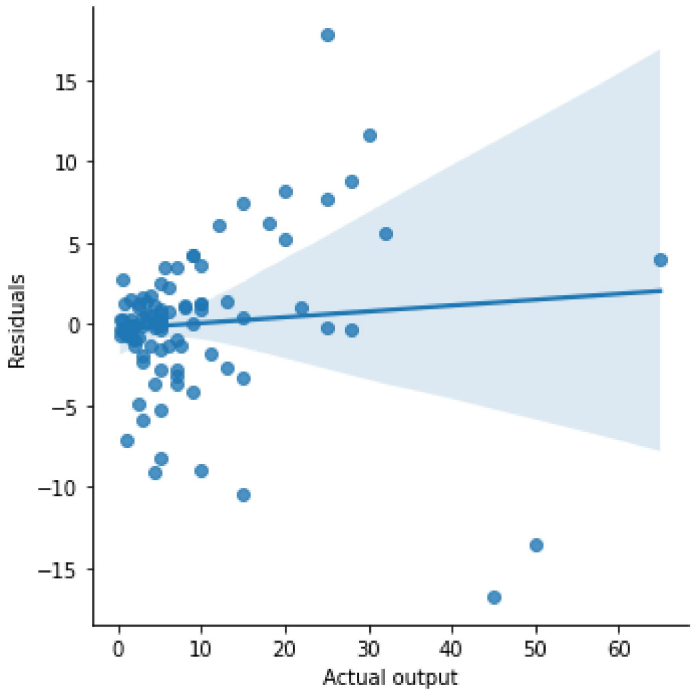

```
R2 Score:  0.8545362080019921
RMSE value:  4.196264616479616
Training R2 Score:  0.9699236829125817
```



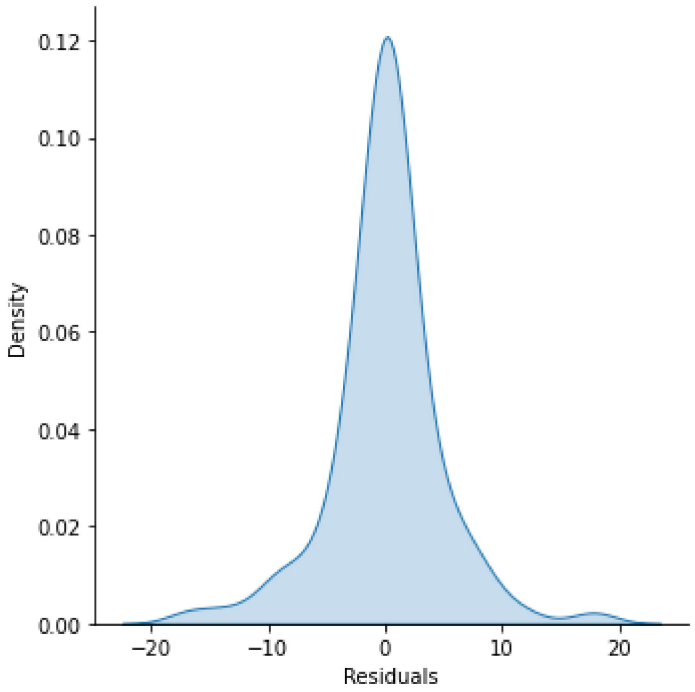
In [42]:
```
#For GradientBoosted Regression
cv = RepeatedKFold(10,3,2)
estimator = GradientBoostingRegressor()
parameters={
     "loss":['ls', 'lad', 'huber'],
     "learning_rate":np.linspace(0,1,11)[1:],
     "n_estimators":[100,250,500],
     "max_features":[None,"auto", "sqrt", "log2"],
     "criterion":["mse", "friedman_mse", "mae"]
}
gridsearch = RandomizedSearchCV(estimator,parameters,scoring='r2',cv=cv)
model=gridsearch.fit(X_train,y_train)

bestparams = model.best_params_
print("Best Parameters: ",bestparams)
```

```
Best Parameters:  {'n_estimators': 500, 'max_features': 'sqrt', 'loss': 'ls', 'learn
ing_rate': 0.1, 'criterion': 'friedman_mse'}
```

In [43]:
```
model = GradientBoostingRegressor().set_params(**bestparams)
model.fit(X_train,y_train)
model_pred = model.predict(X_test)
model_df=pd.DataFrame(list(zip(y_test,model_pred,y_test-model_pred)),columns=['Actual output','Predicted output',"Residuals"])
sns.lmplot(x='Actual output',y='Residuals',data=model_df)
plt.show()
sns.displot(model_df,x='Residuals',kind='kde',fill=True)
print("R2 Score: ",r2(y_test,model_pred))
print("RMSE value: ",mse(y_test,model_pred)**0.5)
y_pred=model.predict(X_train)
print("Training R2 Score: ",r2(y_train,y_pred))
```




```
R2 Score:  0.8189746234091625
RMSE value:  4.681177882847442
Training R2 Score:  0.9962620792466541
```



Genetic Algorithm

In [19]:

```
def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)


def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance) - 1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(min(k, len(distances))):
        neighbors.append(distances[x][0])
    return neighbors


def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True
)
    return sortedVotes[0][0]


def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct = correct+1
    return (correct / float(len(testSet))) * 100.0


def kNN(k, testSet, trainingSet):
    # generate predictions
    predictions = []
    for x in range(len(testSet)):
        neighbors = getNeighbors(trainingSet, testSet[x], k)
        result = getResponse(neighbors)
        predictions.append(result)
    accuracy = getAccuracy(testSet, predictions)
    return accuracy


def crossover(c1,c2):

    male = bin(c1)[2:]
    female = bin(c2)[2:]
    length = max(len(male), len(female))

    if length % 2 == 1:length = length + 1

    male = "0"*(length-len(male)) + male
    female = "0"*(length-len(male)) + female

    child = []
    half = length//2
```

```
    male1 = male[:half]
    male2 = male[half:]

    female1 = female[:half]
    female2 = female[half:]

    child.append(int(male1 + female2,2))
    child.append(int(female1 + male2,2))
    return child

# prepare data
trainingSet = X_train.to_numpy()
testSet = X_test.to_numpy()
split = 0.67
accResult = [[]]
chromosome = 10
population = [rand.randint(1, 100) for x in range(chromosome)]

for i in tqdm(population):
    accResult.append([i, kNN(i, testSet, trainingSet)])

del accResult[0]

for i in range(200):
        status_one = True
        status_zero = True
        accResult = sorted(accResult, key=lambda x: x[1], reverse=True)
        newChromosome = crossover(accResult[0][0], accResult[1][0])
        for i in accResult:
            if newChromosome[0] == i[0]:
                status_zero = False;
            if newChromosome[1] == i[0]:
                status_one = False
        if status_zero:
            accResult.append([newChromosome[0], kNN(newChromosome[0], testSet, traini
ngSet)])
        if status_one:
            accResult.append([newChromosome[1], kNN(newChromosome[1], testSet, traini
ngSet)])

accResult = sorted(accResult, key=lambda x: x[1],reverse=True)
print("accuracy: ",accResult[0][1])
```

```
c:\users\abhin\appdata\local\programs\python\python37\lib\site-packages\ipykernel_la
uncher.py:83: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`


accuracy:  90.32258064516128
```

Pickling Best Model

In [25]:

```
import pickle
params = {'n_estimators': 200, 'max_features': 'sqrt', 'max_depth': None, 'criterion'
: 'mae'}
model = RandomForestRegressor().set_params(**params)
model.fit(X_train,y_train)
with open("RandomForest.pkl","wb") as file:
    pickle.dump(model,file)
```

In [26]:
```
with open("RandomForest.pkl","rb") as file:
    load = pickle.load(file)
load.score(X_test,y_test)
```

Out[26]:
```
0.850998221487706
```